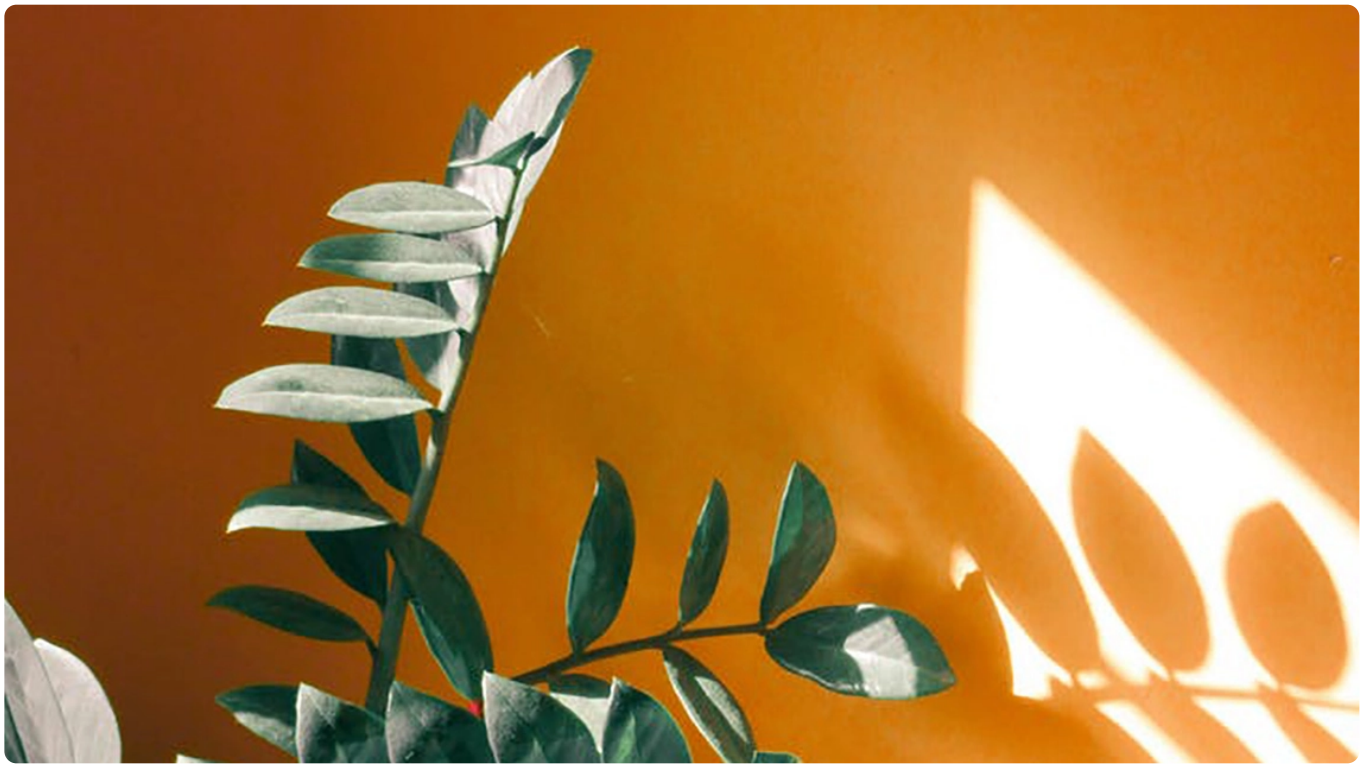
13｜优化番外篇：Vertx介绍及快速入门

2021-10-25 佘志东

《手把手带你搭建秒杀系统》 课程介绍 >



讲述：佘志东
时长 14:10 大小 12.98M

你好，我是志东，欢迎和我一起从零打造秒杀系统。

经过前面课程的学习，我们知道 Nginx 和 Tomcat 都可以做网关服务，并且从理论出发做了分析比对，也从实践上做了相应服务的开发，那么今天我们将学习一款优秀的、可开发网关服务的技术，即 Vertx。该技术的总体性能要优于 Tomcat，但弱于 Nginx。其在国内的普及度相对国外来说还是比较低的，但已经有些公司开始尝试使用了，比如京东的 PC 商详页服务、秒杀 Web 服务都是用它来开发的，并且线上实际效果也很不错。

接下来我们将对 Vertx 做个简单的介绍，并实际搭建一个 Vertx 项目，来替换 demo-[illegible] 的角色，重新构造秒杀 Web 系统的一环。

Vertx 简介

我们先了解一下 Vertx 可以用来干什么，这样我们才能在已知的技术栈中找出一个和其相对应的技术来帮助理解。

首先它可以开发 Web 服务，这点 Nginx、Tomcat 也能做。

其次它也有点像 Spring，它提供了完整的生态，包括 Vertx Web、Vertx Core、Vertx Unit 等模块，但它和 spring 也不是互斥关系，我们待会搭建的项目中就会使用 Vertx+Spring，它们可以配合使用。

然后它还提供了很多其他的能力，比如 EventBus 的消息机制，实现服务间通信，同时它还可以支持多种语言的开发。

所以我们很难直接给它做一个精确的定义，但其总体的设计思路就是异步非阻塞。这里不仅仅是针对我们熟悉的一些请求消息的读写，还包括下游服务的调用，甚至是数据库的读写都可以实现异步，支持你去编写全链路的非阻塞代码。这样可以使用有限的线程，来充分利用机器的 CPU 资源，当然这并不能缩短业务接口的执行时间，这点你得清楚。

那么从以上这几个方面来看的话，我们其实很难用一节课把 Vertx 技术介绍全面，所以这里我们定个更加好实现的小目标，那就是用 Vertx+Spring 来开发一个 Web 服务。你可以在这个过程中，和 Nginx 还有 Tomcat 做个简单的对比。

Vertx 对比 Nginx和Tomcat

首先，Nginx 由于多进程、单线程、多协程的模式，其占用资源更少，并发处理能力更强，再加上其事件驱动、异步非阻塞等特性，总体性能也是三者中最好的，并且比 Vertx 和 Tomcat 要强出很多。

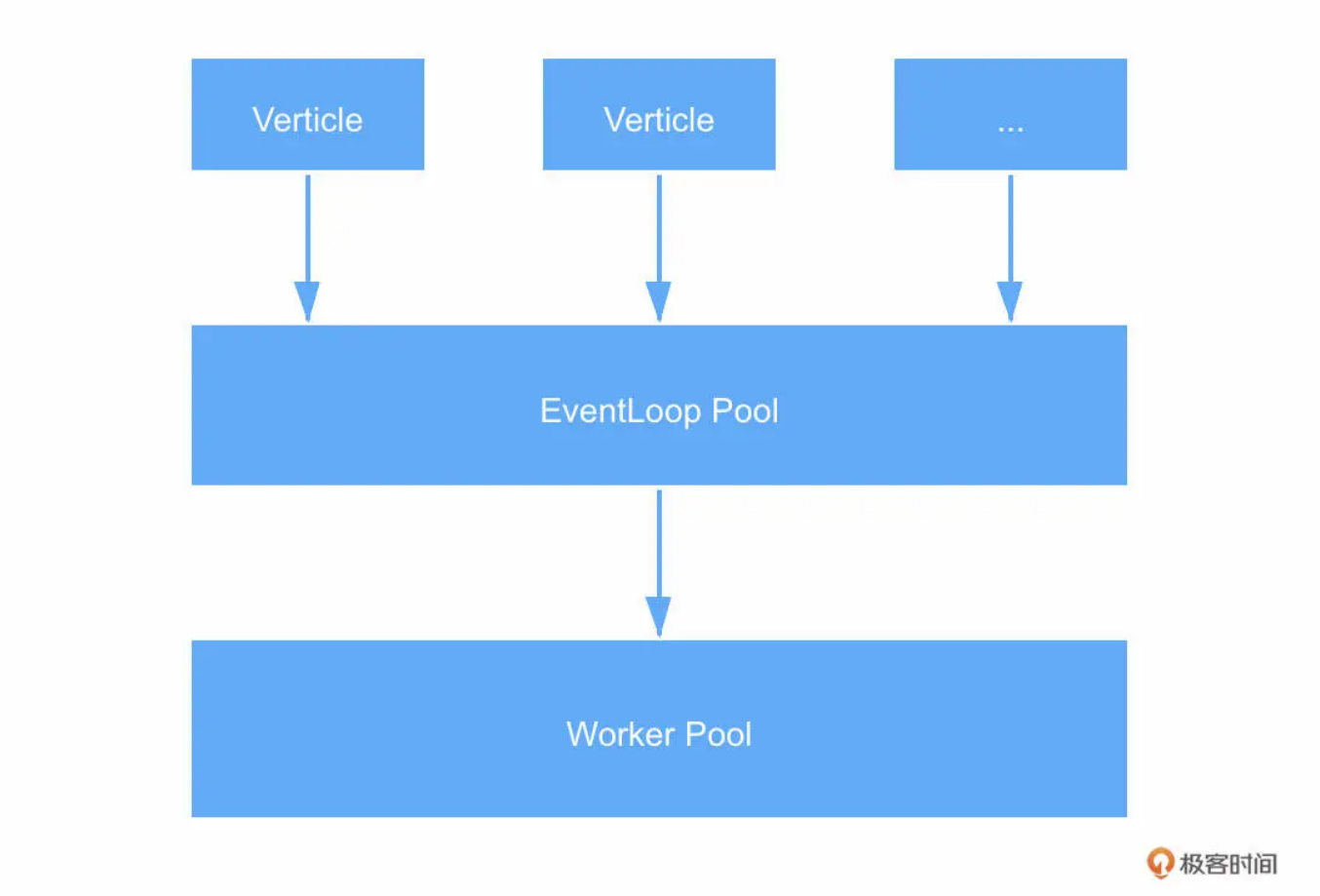
Nginx 还可以支持使用 Lua、C 等语言做业务开发，但大部分以 Java 为主的公司能够支持的 SDK、中间件等相对较少，所以一般都是用它来做反向代理和负载均衡器。因为秒杀的特殊业务场景，故而我们启用了 Nginx 来做前置网关。

Tomcat 的话，大部分公司都用它来部署服务，不管是 HTTP 服务还是 RPC 服务。虽然底层线程模型一直在迭代更新，新的 NIO 模式也大大提高了并发处理能力，但由于其历史更久、内部结构更重，所以整体性能也是三者中最差的，但好在普及度高，绝大部分的公司

技术栈里都有它。所以在秒杀系统中我们暂时只用其来启动 RPC 服务，不再用它来部署 Web 服务了。

最后是 Vertx，就像上面介绍的那样，其在诞生之初，就是以纯异步化的思想来设计的。其内部提供的几乎所有接口都支持异步，并且所有的操作都是基于事件来实现程序的非阻塞运行。再加上它的网络通信框架集成了 Netty，其内部的零拷贝机制、基于内存池的缓冲区重用机制，还有高性能的序列化框架，以及它本身的 Multi-Reactor 模式，总结来看它的性能是要优于 Tomcat 的。但由于它仍使用线程池来处理业务请求，所以整体性能是弱于 Nginx 的。

它的底层线程模型，大概是这样的，如下图所示：



简单来说，Vertx 在启动时，可以部署多个 Verticle 实例，每个 Verticle 实例都是 Actor 模型的一种具体实现（其内部定义好了整个流程的处理顺序，当然我们可以针对可扩展项做自定义开发）。

它接受请求，然后将其封装成事件，并交给 EventLoop 线程来处理。当然在 EventLoop 线程中，我们也可以决定是否将后续事件交给 Worker 线程来处理，比如像调用外部 RPC

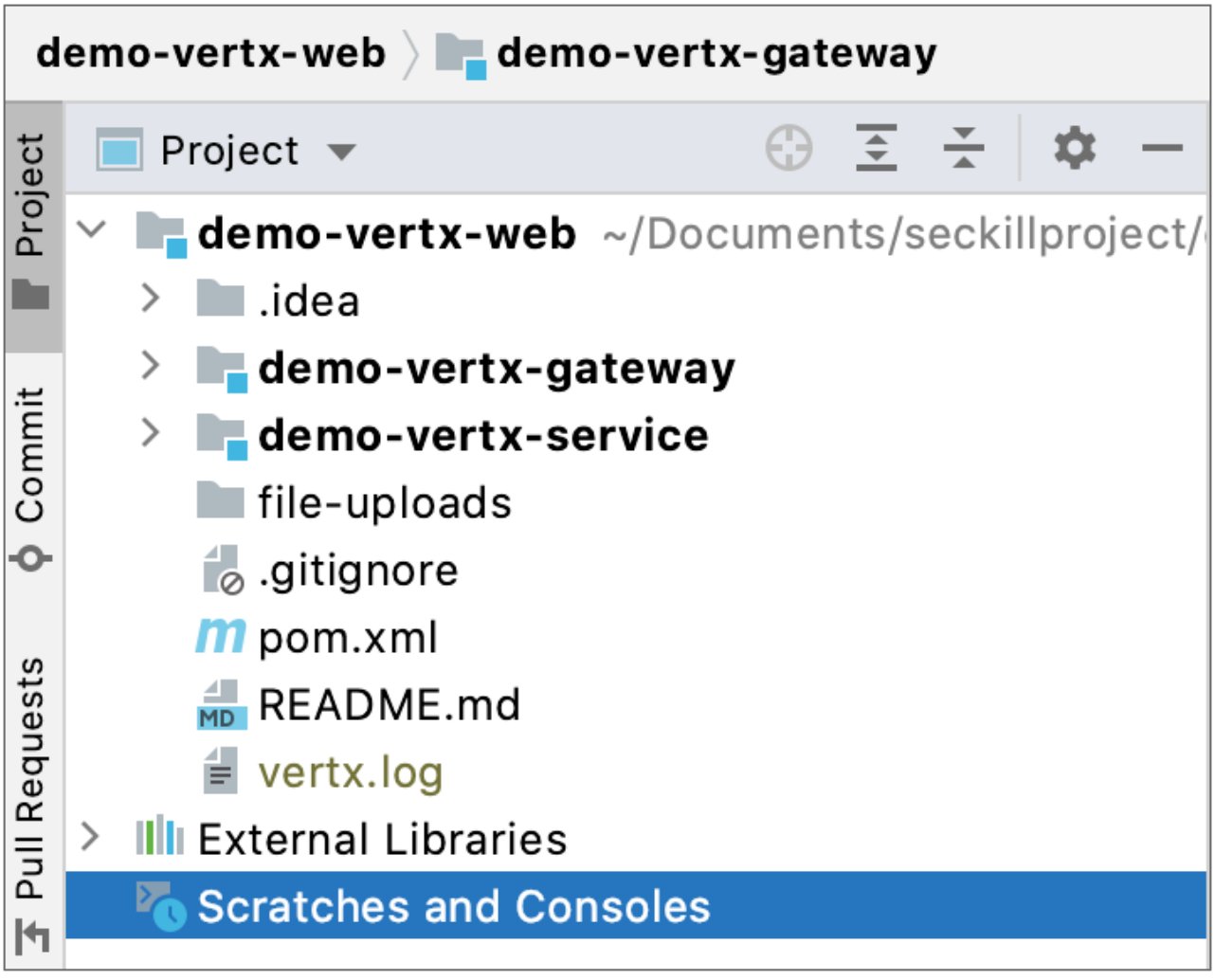
接口等 IO 操作。

这里有个黄金原则，即在 EventLoop 线程里运行的程序一定不能阻塞该线程，否则这将直接影响其处理后续请求的效率。正是因为其有着不错的性能表现，所以在秒杀系统中，我们使用它代替了 demo-web，来开发新的 Web 服务。

那在了解了 Vertx 的一些基本概念之后，咱们来实际搭建一个 Vertx Web 项目，使其具备 demo-web 的能力，同时上下游系统 demo-nginx 和 demo-support 不用做任何的改动。下面咱们就正式开始吧！

Vertx 项目搭建

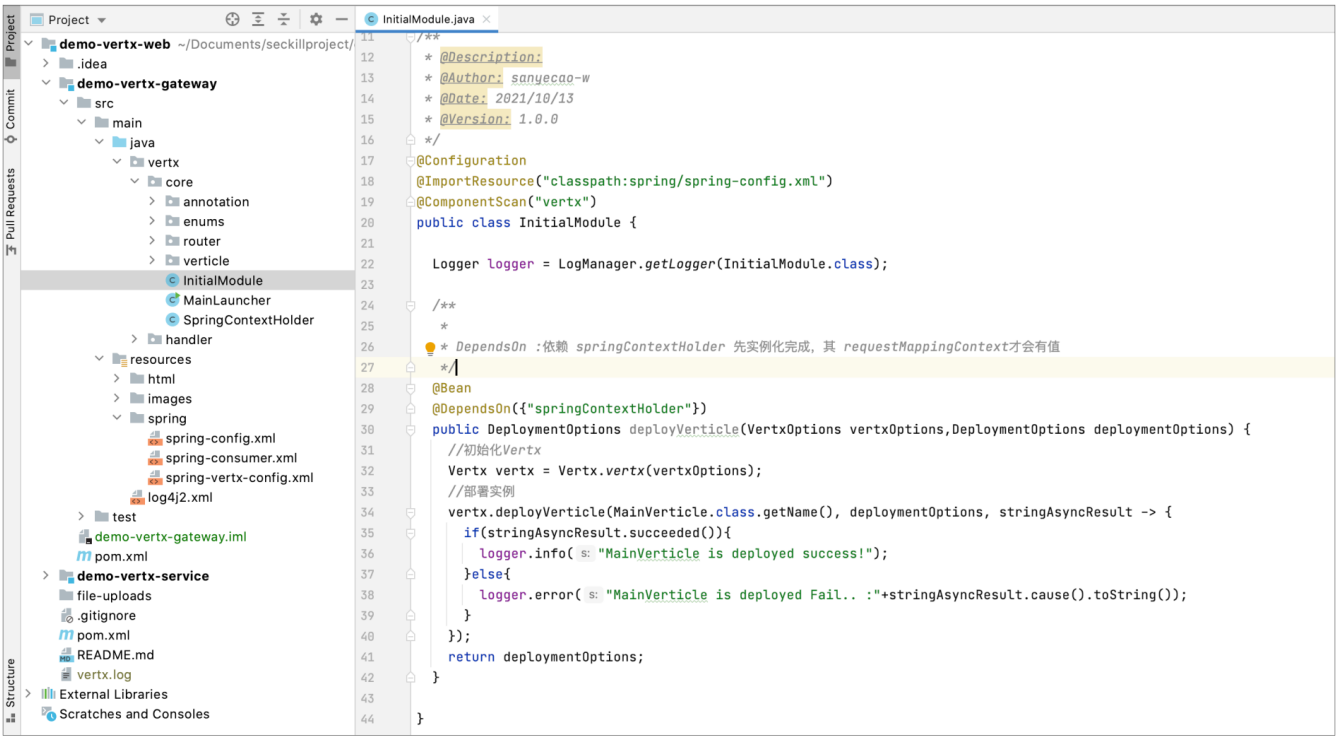
第一步：新建一个项目 demo-vertx-web，并包含两个 module，整体项目结构和 demo-web 类似，如下图所示：



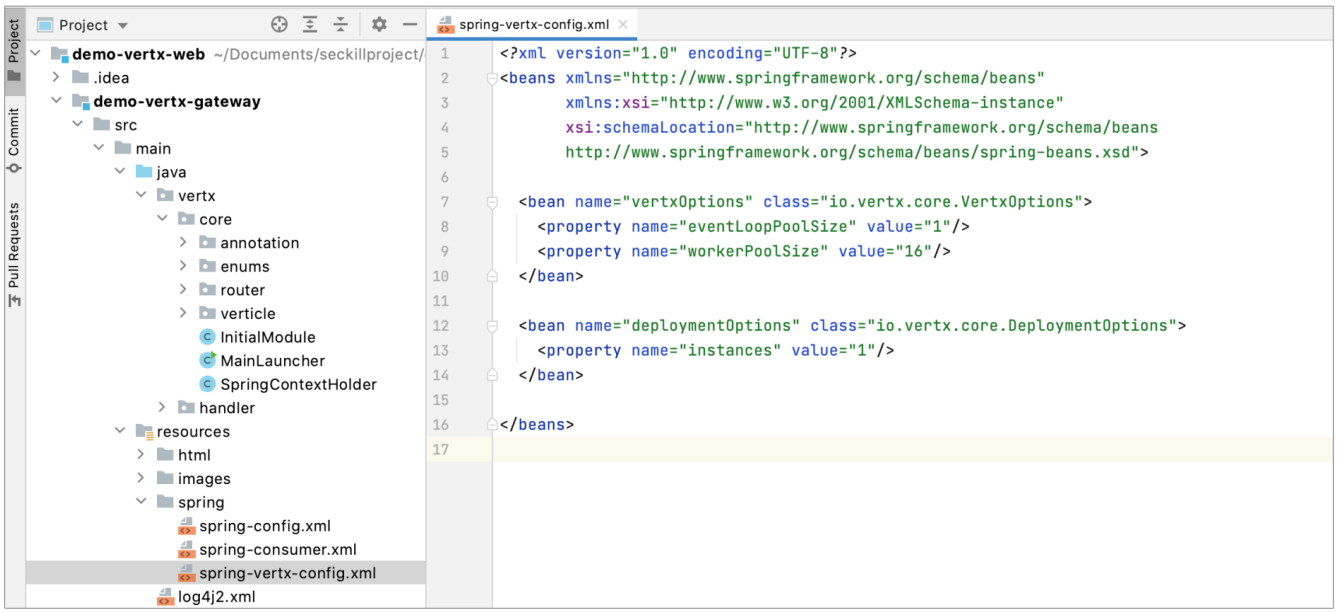
其中 demo-vertx-gateway 模块用来做一些文件的配置，并且也是用来打包部署的，demo-vertx-service 主要用来做业务聚合，这块的代码到时候可以直接将 demo-web 中的 service 模块代码拷过来即可。所以我们主要的改动点也都是在 demo-vertx-gateway 这一块。

第二步：由于没了 Tomcat 来启动服务，所以我们需要自己写个 main 方法，来加载对应的配置，完成项目的启动，由于我们仍然使用 Spring 来管理类，所以这里也通过 Spring 提供的机制来完成对应配置的加载。如下图所示：

那我们继续看下 InitialModule 类都配置了些什么：



这里首先是导入 Spring 的配置文件 spring-config，同时配置 Spring 的包扫描路径，其次是初始化 Vertx，然后开始部署 Verticle 实例，这里 Vertx 的初始化配置和部署实例的配置如下：



vertxOptions 主要配置了两个属性，即 EventLoop 线程池大小和 Worker 线程池大小，而 deploymentOptions 主要配置了部署的实例数。

这里要注意一下，一个 Verticle 实例会被指定到一个 EventLoop 线程处理，一个 EventLoop 线程可以处理多个 Verticle 实例，所以这里的 EventLoop 线程数最好和 CPU 核数相等，而部署的实例数，最好小于等于 EventLoop 线程数，这样可以确保一个线程只对应处理一个 Verticle 实例。比如你是 8 核机器，那就配置 8 个 EventLoop 线程和 8 个 Verticle 实例。

第三步：从上面可以看到，我们部署的实例是 MainVerticle，在这里我们可以编排请求的处理流程，先看下代码，如下图所示：

MainVerticle.java

```
17    */
18   public class MainVerticle extends AbstractVerticle {
19     @Override
20     public void start(Promise<Void> startPromise) {
21       //创建router
22       Router router = Router.router(vertx);
23
24       //1.给router配置通用handler
25       configCommonHandler(router);
26
27       //2.配置router-业务URL与处理方法
28       new RouterHelper(router).configRouter();
29
30       //3.静态资源配置，设置启用缓存
31       StaticHandler staticHandler = StaticHandler.create().setCachingEnabled(true);
32       staticHandler.setWebRoot("demo-vertx-gateway/src/main/resources");
33
34       //3.1 静态HTML，正则匹配
35       router.routeWithRegex( s: ".*\\.html").blockingHandler(staticHandler,  b: false);
36
37       //3.2 静态JPG，正则匹配（静态资源上到CDN后，则不再需要相关配置）
38       router.routeWithRegex( s: ".*\\.jpg").blockingHandler(staticHandler,  b: false);
39
40       //4.设置Http服务配置
41       HttpServerOptions serverOptions = new HttpServerOptions();
42       //连接超时时间（保持长连接）
43       serverOptions.setIdleTimeout(20000).setIdleTimeoutUnit(TimeUnit.MILLISECONDS);
44
45       //5.启动服务，设置监听端口号
46       vertx.createHttpServer().requestHandler(router).listen( i: 8080, http -> {
47         if (http.succeeded()) {
48           startPromise.complete();
49         } else {
50           startPromise.fail(http.cause());
51         }
52       });
53     }
```

这里我们首先是创建了一个 Router，即路由，这个类主要是用来设置请求的处理逻辑，包括哪个 URL 请求由哪个类的方法来处理，是让 EventLoop 线程来执行，还是让 Worker 线程来执行，处理的过程是什么样的等等。

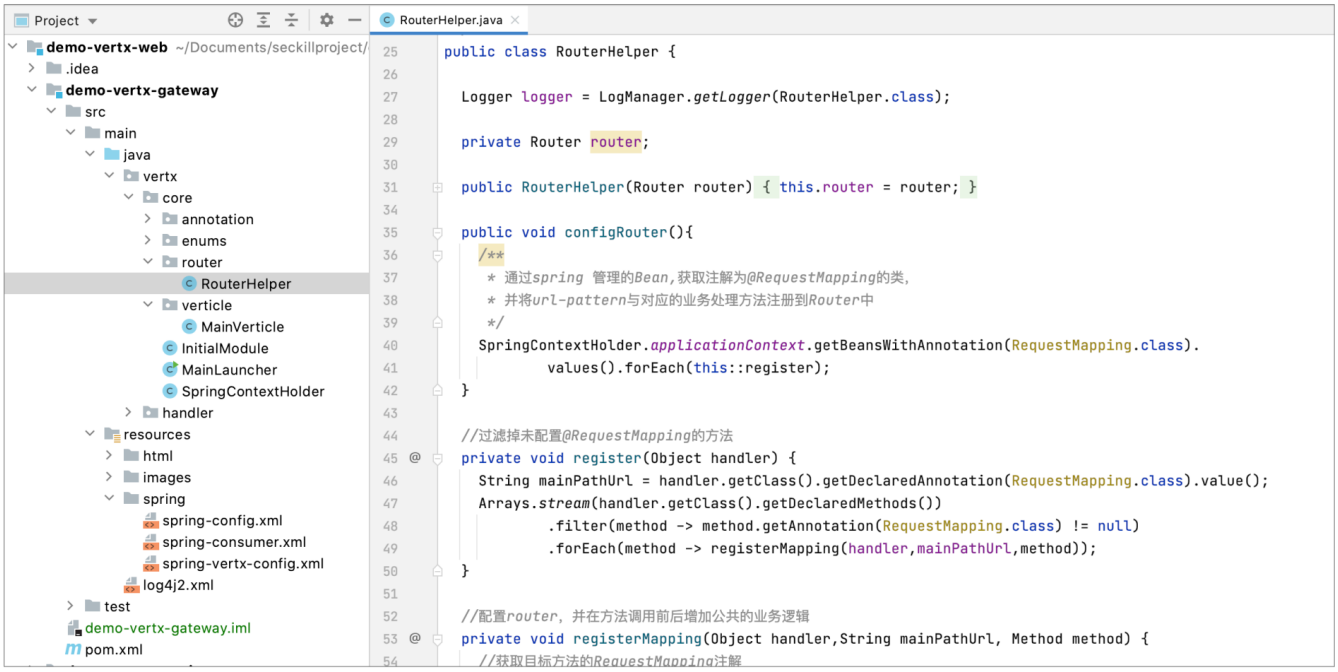
其次我们就按上面说的对 Router 进行配置了。这里分了两种情况，一种是处理动态请求的，一种是处理静态资源请求的。动态请求的配置咱们先放一下，等会细说，这里先说下配置静态资源请求的情况。

这里统一都是由 StaticHandler 来处理，同时设置了启用静态资源的缓存，这样就不用每次都读文件了，大大地提高了响应的速度。

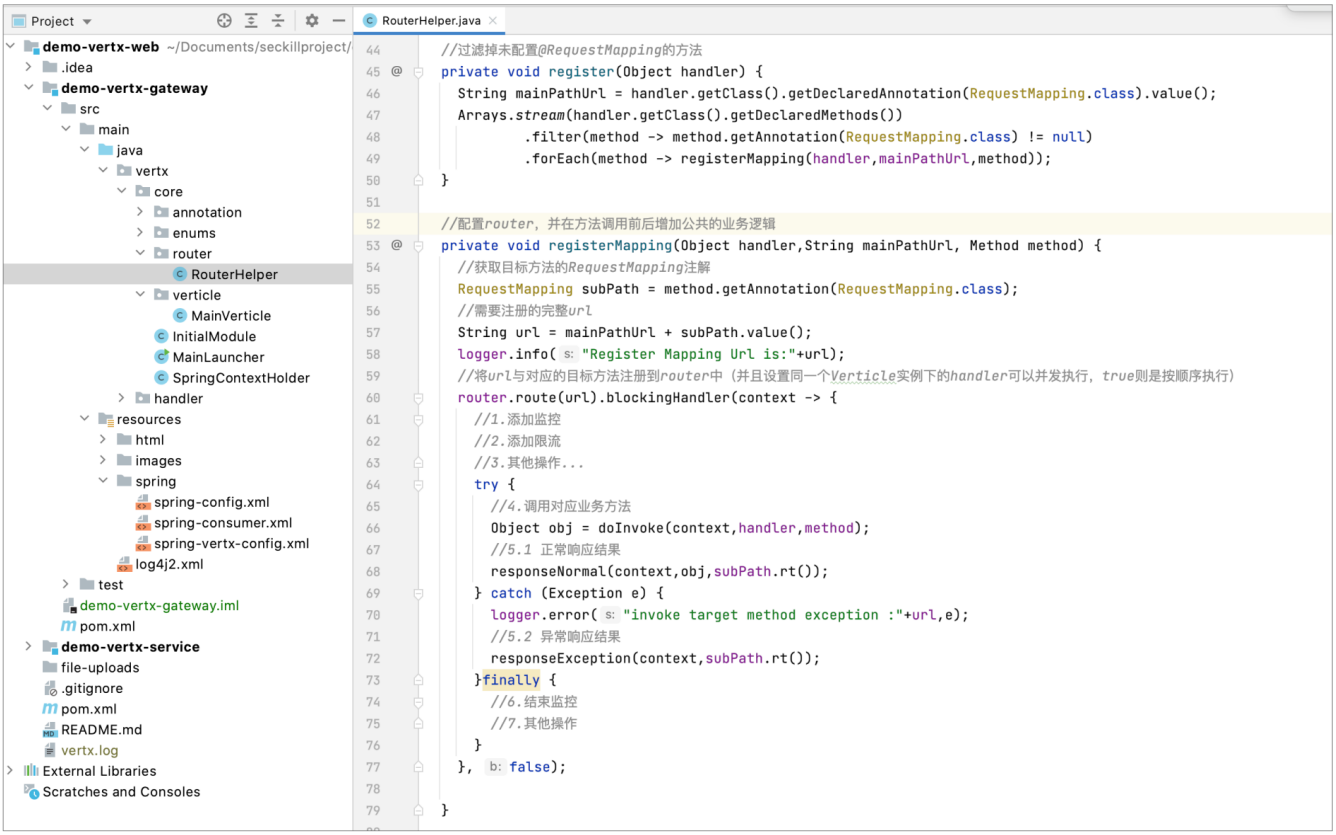
StaticHandler 处理静态资源的逻辑是直接根据你请求的 URL，然后加上配置的 WebRoot，来读取文件。所以比如我们想获取结算页 H5，我们的请求 URL 是 /settlement/page，这个时候需要我们 reroute 一下，重新路由到对应要返回的 HTML 文件路径，这块也会在下面说到。

设置好路由 Router 之后，我们就可以配置 HTTP 服务的启动配置了，包括各种 HTTP 相关的设置，如连接超时配置、端口号设置等。

第四步：这块再详细说下动态路由的配置，我们是通过 RouterHelper 来做相应配置的。这块的主要功能是将请求 URL 与对应的处理方法做绑定，并注册到 Router，这样请求进来后，就知道该执行哪段业务逻辑了。



这里首先是从 Spring 管理的 Bean 中，找出添加了我们自定义注解 RequestMapping 的类，然后遍历，将请求 URL 与对应的类和方法注册到 Router 中，具体的注册方法如下：



这里我们指定了业务逻辑由 Worker 线程来执行，那是怎么指定的呢？

我们是通过 Router 的 blockingHandler() 方法来指定的。如果想在 EventLoop 线程里执行，则使用 route.handler() 方法。关于如何合理分配任务，我们有个简单的基本原则，比如你想每秒处理 1000 个请求，那么理论上每个请求的处理时间不能超过 1ms，如果超过了 1ms，那就不应该放到 EventLoop 线程里执行，否则会影响后续请求的处理，所以一般为了简单起见，我们都是将业务逻辑直接分配给 Worker 线程来处理。

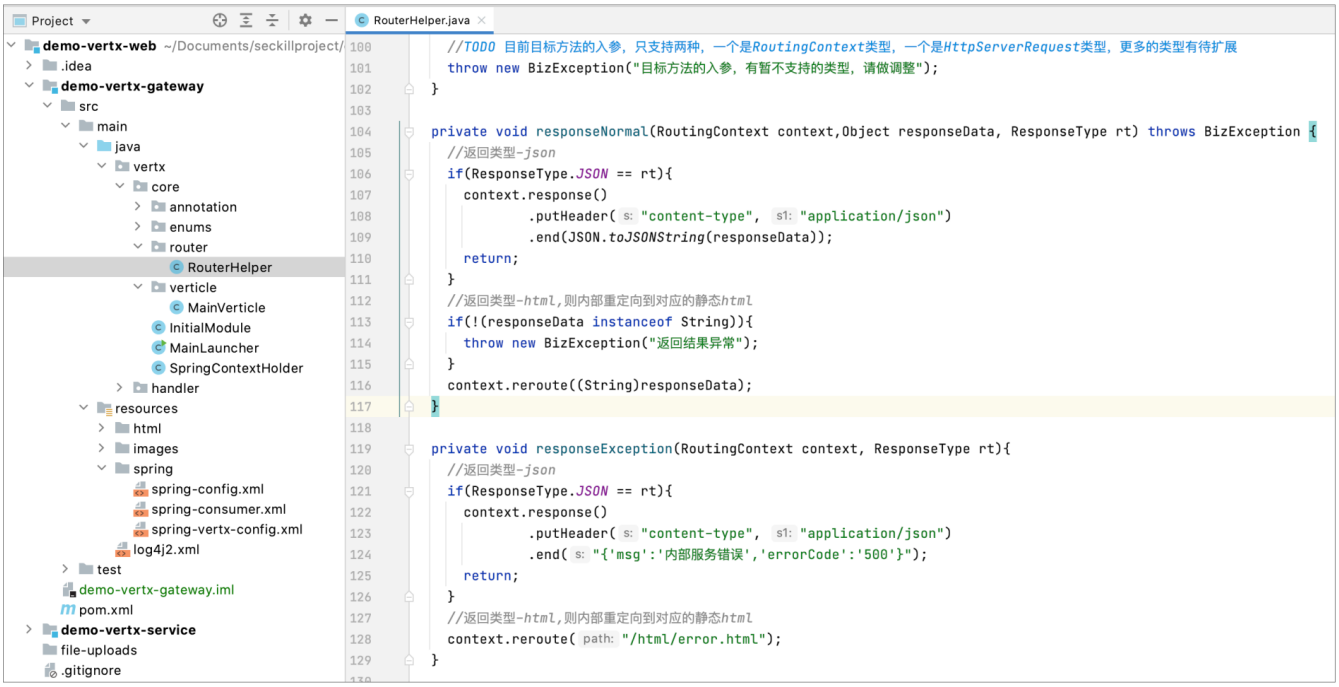
另外在 blockingHandler 中，我们可以在执行目标方法之前，增加像监控、限流之类的公共逻辑，有点像拦截器的概念。然后执行目标方法的的调用，这里是通过 Java 的反射来做的。

这里目标方法的入参，暂时只适配了 2 种，一个是 RoutingContext，另一个是 HttpServerRequest。

这是 Vertx 的一个弊端，它没有提供直接的功能，无法像 SpringMVC 一样方便，能够支持在 Controller 的方法入参里直接定义我们想要的变量来接受入参的值，得需要我们自己去扩展。所以这里为了简便，我们直接先将原始的 RoutingContext 或 HttpServerRequest 往下传，其中 RoutingContext 是一次请求的上下文，HttpServerRequest 是将 HTTP 请求的相关信息都封装进去了，包括入参、Header 等。

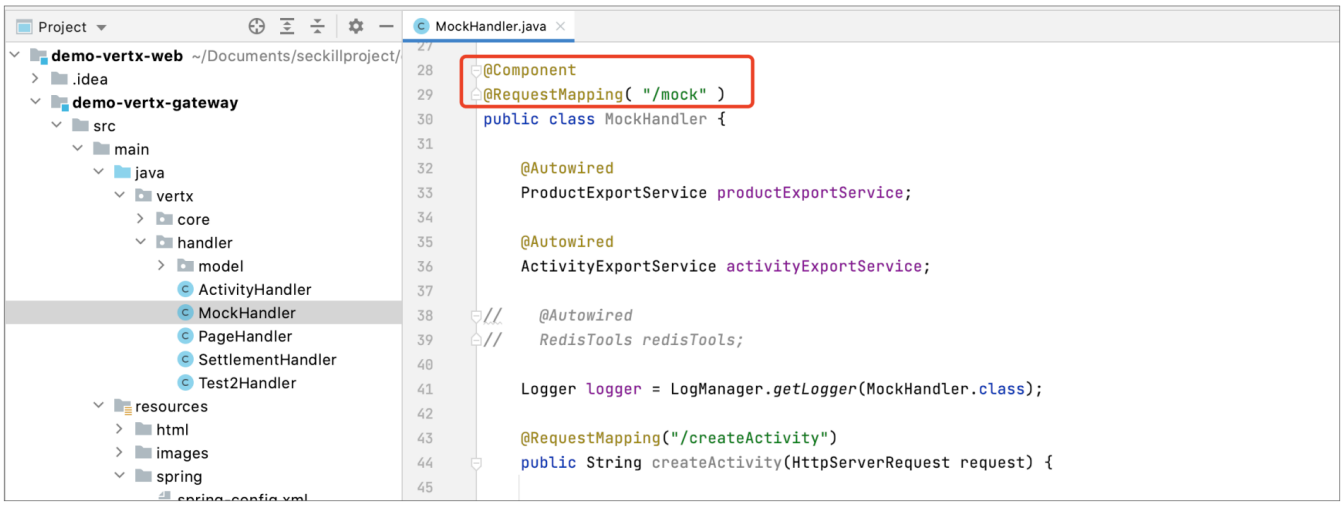
另外这里要重点关注一下，blockingHandler() 方法的第二个入参，可以设置 Hander 是顺序执行还是并发执行。这个是指同一个 Verticle 实例下，同一个适配 URL，默认是 true，但在高并发场景下，需要设置成 false，否则吞吐量上不去。

最后，在执行完目标方法之后，需要将响应结果返回。我们根据响应类型，也做了区分处理，目前支持两种，一个是 JSON 返回，另一个是 HTML 返回。对于 HTML 的返回，我们 reroute 到了真实的静态文件路径，如下图所示：



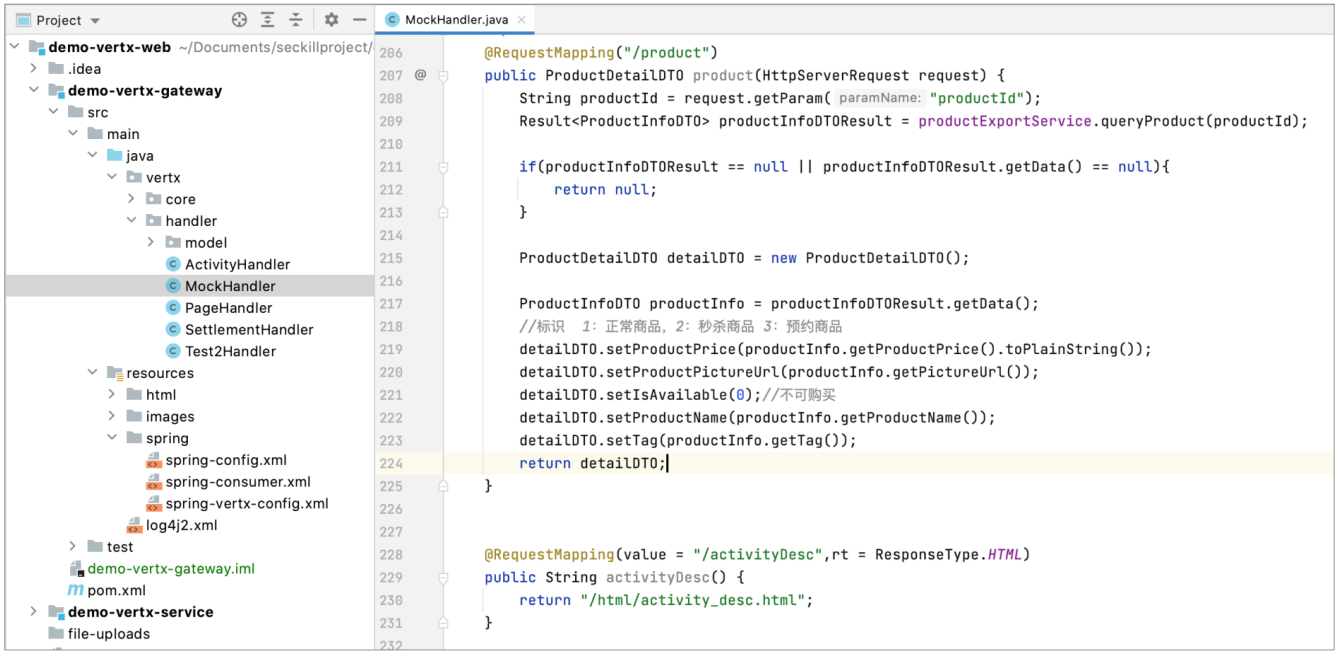
并且在异常时，也做同样的区分处理。

第五步：到这里，一些核心逻辑的编写就完成了。下面再看一个业务 Handler 的示例，为了让你适应 SpringMVC 的使用习惯，所以这里的写法风格上也是类似的：



这里要记得加上 Spring 的 @Component 注解，这样 Spring 才能将其管理起来。同时也要加上自定义的 @RequestMapping 注解，这样才能被我们自己的逻辑扫到。

那下面再展示下两种响应类型的写法，如下图所示：



一个是默认返回 JSON，另一个是返回 HTML（真实的 HTML 文件路径），HTML 的返回需要指定 RT 的类型。

其他的如 Dubbo 相关配置，都是和 demo-web 一样的，剩下的我们只需要将 demo-web 中的业务代码拷过来即可。唯一有个变动的点，就是在活动查询接口有个请求跨域问题，需要在 Nginx 中设置响应结果的 Header，其他的都没有变化。

那这样，咱们的 demo-vertx-web 项目就算搭建完成了，之后可以按照以前的测试方式，从商详页进入到结算页然后完成下单，测试整个秒杀流程，这里就不再复述了。

总结

这节课我们初步认识了 Vertx，这是一种以异步化思想来设计的技术。其内部提供的几乎所有接口都是异步的，并且像传统的 RPC 调用、数据库读写等阻塞操作，Vertx 也都提供了异步化、非阻塞的解决方案。

这和我们传统的开发方式有所不同，你可以借助 Vertx 习惯这种新的开发思想。当然异步化编程势必会增加代码的复杂度，这也是其弊端。而对于开发 Web 服务这块，我们横向对比分析了 Vertx 与 Nginx、Tomcat 的优劣势，其性能介于 Tomcat 与 Nginx 之间，而这三种技术都有其自身的优缺点，并且我们在秒杀系统中都有使用到。

像 Nginx，性能最优，我们用其来做前置网关，直面大流量的冲击；Vertx 性能次之，所以我们用其来开发业务 Web 服务；Tomcat 使用起来最方便也最普及，可以用来发布

RPC 服务，或者是像 ERP 这种小流量的 Web 服务。

同时在今天这节课中，我们也用了较多的篇幅，来讲解 Vertx 开发 Web 服务的详细过程，帮助你更直观地学习 Vertx，并投入到实际的使用中。

虽然如此，但今天介绍的也仅仅是 Vertx 的冰山一角，这里只是起到抛砖引玉的作用，Vertx 所具备的能力绝对不止这些。如果你感兴趣，可以自己做进一步的学习，当然更多的相关开发使用，也可以关注我的 GitHub，我们一起交流学习。

思考题

这节课我们实际开发的 Vertx 项目，处理业务逻辑这块，仍然使用的是 Worker 线程池，这块和 Tomcat 没有太大区别。如果我们使用 Vertx 提供的异步化 API，该如何改造现有代码呢？又会带来怎样的影响？

期待你的思考，欢迎你来评论区和我讨论问题，交流经验！我们下节课再见。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 0 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 12 | 高性能优化：单机Java极致优化

精选留言 (1)

写留言

qinsi
2021-10-25

多年以前用过Vert.x 2，开发上的体验接近Node.js（毕竟原本就叫Node.x）。缺点是招不到人，招进来的Java开发都只会写Spring，对着Vert.x代码只会干瞪眼。

1